Did you know …?

15 less known webpack features in 15 minutes

Goals

Goals

- Overview over less known features
- Short
- Learn something new
- Code Samples

Non-Goals

- Detailed How-to-Use the features
- Well formatted
- Amazing Animations

import()

- webpack 3 added a new syntax to load modules on demand.
- It‘s covered by spec. Native support by Chrome and Safari.

Usage:

```
import("./module.js").then(module => {
    console.log(module.default);
});
// or in a async function:
const module = await import("./module.js");
console.log(module.default);
```

import() magic comment

- A magic comment allows to provide (webpack-specific) extra information.
    - Chunk name, context mode (webpack 3)
    - Include, Exclude files (webpack 4)

```
// for a single file
import(/* webpackChunkName: "database" */ "./database.js");

// with expression: route-1, route-2, route-3, ...
import(/* webpackChunkName: "route" */ `./routes/${name}.js`);

// with expression: route-dashboard, route-login, ...
import(/* webpackChunkName: "route-[request]" */ `./routes/${name}.js`);
```

import(/* webpackMode */)

```
// create one chunk per module (default)
import(/* webpackMode: "lazy" */ `./routes/${name}.js`);

// create one chunk for all modules
import(/* webpackMode: "lazy-once" */ `./routes/${name}.js`);

// create no chunk, include modules in referring chunk, no network request
import(/* webpackMode: "eager" */ `./routes/${name}.js`);

// don’t include modules, reject promise if module is not available at runtime
import(/* webpackMode: "weak" */ `./routes/${name}.js`);
```

resolve.alias

- You can redirect module requests to your wishes
- Use cases: Drop-in-replacements, wrong/missing main field, application root

```
resolve: { alias: {
    // drop-in
    "underscore": "lodash",
    "lodash": "lodash-es",

    // missing main field
    "broken-module": "broken-module/dist/broken-module.js",

    // application root
    "app": path.resolve(__dirname, "src"), // require("app/abc")
    "~": path.resolve(__dirname, "src") // require(“~/abc")
} }
```

resolve.modules

- You can configurate where webpack looks for modules
- Use cases: Directory with own modules

```
resolve: {
    modules: [
        // Add you own directory
        path.resolve(__dirname, "src/modules"),
        // Look in node_modules too
        "node_modules"
        // Priority by order
    ]
}
```

resolve.plugins

- In addition to webpack plugins, you could also add resolver plugins
- Use cases
  - Using weird legacy resolving algorithm while migrating
  - Special resolving rules that make sense to you
  - Automatic installation of missing npm modules
- But: Try to stick with defaults for best combatiblity with other tools

```
resolve: {
    plugins: [
        new MyCustomResolverPlugin(options)
    ]
}
```

output.library

- webpack offers a lot of options to compile as library
- CommonJS, AMD, global var, UMD

```
output: {
    library: "MyLib",
    libraryTarget: "umd"
}
```

- Also affected by the `externals` option

devtool: eval

- The SourceMap devtools are well known, but there is an alternative.
- Eval-Devtool
    - only displays the generated code
    - Separated by modules
- Much faster than other devtools

```
devtool: "eval"
```

SourceMapDevToolPlugin

- Instead of `devtool: "source-map"`, you can also use this plugin directly.
- More options, more flexibility

```
plugins: [
    new SourceMapDevToolPlugins({
        test: /\.js$/,
        exclude: /vendor\.js$/
    })
]
```

externals

- Reference modules/variables from the environment.
- Possible types: global, CommonJS (require), AMD (define)

```
externals: [
    // Map by regexp, to the same value
    /^react.*/,
    // Async function for max flexibility
    (request, callback) => { /* ... */ }
]
```

```
externals: {
        // Specify type by prefix
        "underscore": "var _",

        // customize with libaryTarget: "umd"
        "lodash": {
            commonjs: "lodash",
            amd: "lodash",
            root: "_"
        }
    },
    // Async function for max flexibility
    (request, callback) => { /* ... */ }
]
```

electron target

- Compile applications for electron
- webpack knows about electron native modules
- Automatically selects correct way of chunk loading for Code Splitting

```
target: "electron-main"
target: "electron-renderer"
```

node.js target

- Bundle node.js applications
- Use cases: Faster startup, loaders, custom resolving, all webpack features
- But not all modules are compatible → `externals`
- Also great: TODO node-modules externals

```
// loads chunks with require (sync)
target: "node",

// loads chunks with fs and vm (async)
target: "async-node"
```

chunkhash

- Automatically embed hash of chunk for long term caching

```
output: {
    filename: "[chunkhash].js"
}
```

- Final filename can be found in the stats.
- Also great: Use html-webpack-plugin to generate HTML directly

records

- Store module/chunk ids for consistency in a json file.
- File need to be kept between compilations

```
recordsPath: path.resolve(NETWORK_SHARE, "production.json")
```

- But: infrastructure for keeping this file needed

NamedModulesPlugin

- Use the module path as id for the module
    - consitent module ids
- Useful for debugging (recommended in development)
- Bigger bundle size, but gzipped size not much affected
- But: leaks file structure into the bundle file

```
plugins: [
    new webpack.NamedModulesPlugin()
]
```

HashedModulesPlugin

- Similar to the NamedModulesPlugin, but creates hash of name for id.
- No longer leak file structure
- But: increase bundle size

```
plugins: [
    new webpack.HashedModulesPlugin()
]
```

DllPlugin

- Creates a bundle which exposes modules to other bundles.
- Use modules from Dll bundle with DllReferencePlugin
- Use Cases: Build performance, Vendor bundle
- But no Tree Shaking/Scope Hoisting in the Dll bundle

DllPlugin Usage

```
output: {
 library: "dll_[hash]",
 filename: "dll.js"
},
plugins: [
  new webpack.DllPlugin({
    path: path.resolve(__dirname,
                  "manifest.json"),
    name: "dll_[hash]"
  })
]
```

```
plugins: [
  new webpack.DllReferencePlugin({
    manifest:
      require("./manifest.json")
  })
]
```

```
<script src="dll.js"></script>
<script src="app.js"></script>
```

The End, Thanks

15 minutes? Probably not…